Developing a Pinball Game

Results of a practical course at the Chair for Computer Graphics and Multimedia
(RWTH Aachen University, Germany)

Iraklis Dimitriadis* Daniel Hariri† Johannes Klöckner‡ Christopher Tenter§

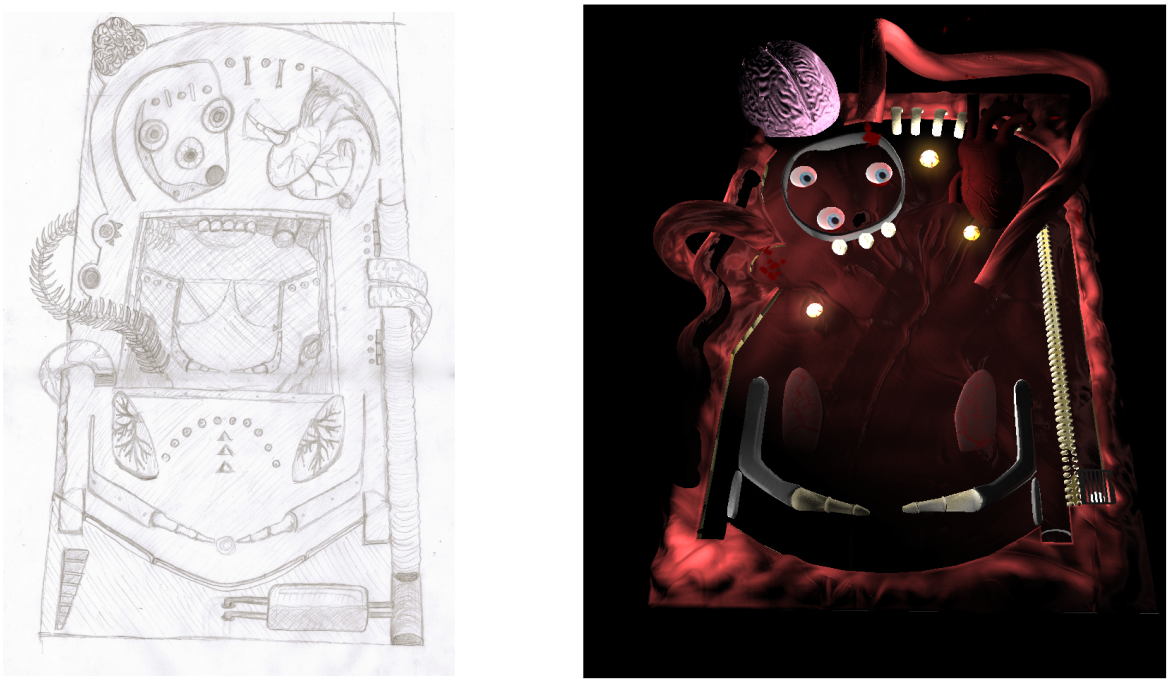


Figure 1: *Concept art vs. final version.*

Abstract

Our idea was to create a 3d pinball game focusing on human anatomy. Emphasis was laid on a more dynamic game environment than in conventional pinball tables. This is mainly achieved by making use of certain graphics effects, that represent a dark hostile, yet playful environment.

The table itself consists of objects like a brain, a beating heart and several other organs. All in all it strongly resembles the internals of a human body.

Keywords: game programming, pinball game

1 Gameplay

The underlying game mechanics do not differ much from other standard pinball games. It is possible to trigger certain actions by shooting the ball against dynamic objects on the table. For instance whenever the ball hits the brain, a randomly chosen mission will be assigned. A typical mission may involve hitting one or multiple objects a given amount of times.

*iraklis.dimitriadis@rwth-aachen.de
†daniel.hariri@rwth-aachen.de
‡johannes.kloeckner@rwth-aachen.de
§christopher.tenter@rwth-aachen.de

In order to implement this kind of interaction we had to design a trigger-event system that keeps track of what should happen, when a certain object was hit.

There are several types of triggers that make up the complete game dynamics. This includes accelerating or bouncing of the ball or simply rewarding the player with a hit score. Another example is the rescue feature whereby the ball is bounced back into the field when it rolls down sideways to the flippers for the first time.

2 Rendering

Fitting to the general idea of this game we chose to implement graphics effects that work well in darker themes. With the forward rendering approach instead of deferred lighting we are able to have different kinds of shading methods for each object. For instance the ball has a transitional surface with a bright color at the top and a darker non-gloomy red at the bottom. This gives the impression, that it actually is one of the light emitters in the scene. However, the forward shading technique forced us to have lots of different versions of similar shaders, whereby only a little, but crucial code is different. In our case the combinatorial problem turned out to be small enough to manage by hand though.

Normal mapping helps to add detail to the game scene, especially in such a dark environment. They enable us to have more bumpy and organic surfaces without increasing the geometric detail. In

fact every visible object is rendered with a normal mapping shader and whenever a normal map was not available, we used a dummy normal map which simulates standard Goraud shading instead.

Subsurface scattering is a technique we implemented to give the heart a more dramatic look. Basically it simulates translucent objects that let a certain amount of light shine through instead of completely blocking it. In our context this means that the pinball illuminates the heart whenever it passes just behind it, giving it a realistically organic look.

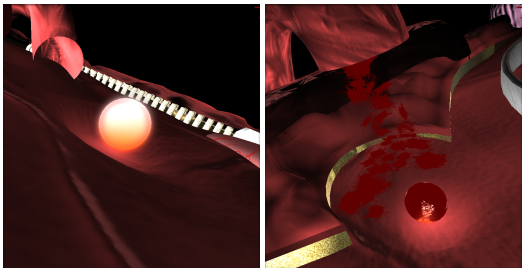
In our game object animations are accomplished via geometry morphing, which is fully performed on the GPU. Morphing allows us to present a more active and dynamic pinball table suitable to the general game theme. Furthermore the constant flow of particles enforce this lively setting, even though the particle system was not utilized to its full extent. Nevertheless our particle system ended up being highly customizable by supporting a wide range of parameters such as speed, size, shape and soft scene intersection [Lorach 2007]. While the constant flow of blood out of veins are modeled via view aligned quads, we experimented with intersecting quads for blood splatter. The latter type copes with the perspective problem where a particle is not in a square shape, but stretched along one axis. Since splatters are aligned along their own traveling direction the viewer often observes them from a nearly perpendicular angle, which should be avoided at all costs.

The particle system is extended by the use of decals. Basically particles can optionally detect collisions with the scene and then, again optionally, spawn decals at the intersection point. Decals were mainly used to create blood stains spawned from blood streams, but in fact they are tightly integrated into the particle system leaving room for further experiments. However, decals proved to be one of the major performance bottlenecks in the later stage of the game. More precisely, the first decal implementation clipped a bounding box in the size of the wanted decal against the scene on the CPU, which performed well for low-poly meshes in the order of about 2500 triangles. Nevertheless, with a more detailed mesh in later stages, the decal system was upgraded to a much faster GPU clipping method as suggested in Gears of War [Smedberg and Wright 2009].

Finally we implemented a post processing glow effect to give light emitters such as lamps and the ball a more realistic appearance.

3 Asset pipeline

Due to the strict time restraint of this course developing a level editor seemed out of place. However, hardcoding the complete level scenegraph quickly became unfeasible, so a new solution had to be found. Given the simple to use interface and accessible scripting support, we turned 3ds Max into our level editor of choice. With our exporter script, the level designer can quickly build and modify the complete game scenegraph without actually changing a single line of game code. This approach payed off rather early in the development process of this course particularly due to the many geometry meshes, physics colliders and trigger objects. It is crucial to understand that the script is used for the level editing aspect only. In fact, we do not export any kind of 3D mesh data with it. Instead it only creates a meta text file containing the scenegraph in a tree structure with typical information like transformation matrices, bounding volumes, physics parameters and references to geometry data files.



(a) (b)

Figure 2: *(a) Glow effect on ball. (b) Particles and decals.*

4 Software Engineering

In the early project stage we carefully discussed requirements and how to create a well designed software architecture. Generally it is is a good practice to continue the model-view-controller pattern based on the given code template. One of the most important aspects was designing the trigger-event system, since many components like trigger object, collision detection and arbitrary event functions have to be taken care of this vital part of the game logic.

Nevertheless, we preferred high functionality and easy integration over good architecture for the rendering code in this project. Particles, decals and glow are examples that work entirely independent from the scene renderer. This modular approach allowed us to add new features without having the risk to run into compatibility problems, which may happen in a stricter designed rendering architecture. Considering the short development time such fatal mistakes can easily avoided by sticking to a more open code design. We even ignored good OOP practices like data isolation for modular classes to allow open access to eventually needed data for the scene rendering routine.

5 Conclusion

In contrast to standard mechanical pinball games, the organic nature of our anatomy pinball allowed us to translate interesting ideas regarding animation of regular pinball objects. A dark atmosphere is created by the use of lack of global illuminating light and matching rendering techniques such as normal mapping and subsurface scattering. Especially the exporter script immensely helped us to cut down development overhead. Unfortunately we were not able to implement all our ideas, however, we managed to convey the impression of a vivid environment.

6 References

References

LORACH, T. 2007. Soft particles.

SMEDBERG, N., AND WRIGHT, D. 2009. Rendering techniques in gears of war 2.